ISIMA 1ère année

vendredi 28 août 2015
Durée : 2 heures

PROGRAMMATION FONCTIONNELLE

Questions préliminaires :

Nota bene : Certaines fonctions demandées dans ces questions préliminaires sont utiles pour la suite du problème.

1°) Ecrire une fonction Scheme `max` ayant comme argument une liste non vide d'entiers `L` et telle que l'évaluation de l'expression `(max L)` retourne l'élément maximum de `L`.
Contrainte imposée : La complexité temporelle de cette fonction doit être linéaire, ce qui signifie que si la liste `L1` est k fois plus longue que la liste `L2`, le temps d'évaluation de l'expression `(max L1)` est de l'ordre de k fois le temps d'évaluation de l'expression `(max L2)`.

2°) Ecrire une fonction Scheme `maxf` ayant comme arguments une fonction `f` d'un seul argument de type T et une liste non vide `L` d'éléments de type T, et telle que l'évaluation de l'expression `(maxf f L)` retourne l'élément x de `L` qui maximise `f`(x).

Exemple : `(maxf (lambda(x) (* x x)) '(-3 1 2)` retourne –3.
Contrainte imposée : Utiliser un `map` afin d'éviter, pour chaque élément x de la liste `L`, d'évaluer plusieurs fois `f`(x).

3°) Ecrire une fonction Scheme `delta` ayant comme argument une liste d'entiers `L` telle que l'évaluation de l'expression `(delta L)` retourne 0 si `L` est vide et retourne

$$\underset{1\le i\le n}{Max}(x_i) - \underset{1\le i\le n}{Min}(x_i)$$

si `L` est non vide, de type $(x_1, ..., x_n)$.
Exemple : `(delta '(1 2 1 2 4 3))` retourne 4 – 1 = 3.
Contrainte imposée : L'évaluation ne doit parcourir la liste `L` qu'une seule fois (la solution consistant à calculer le maximum, puis le minimum, et à faire la différence étant donc proscrite).

Problème :

On donne ici une définition récursive d'un *arbre étiqueté* : un arbre étiqueté A comporte un sommet d'origine r (appelé la *racine*), muni d'une étiquette e (de type quelconque) et un nombre n, éventuellement nul, d'arbres étiquetés $F_1, ..., F_n$ (appelés *sous-arbres fils*, et dont les racines sont appelés les *fils* de r). On peut représenter A par une liste comportant e comme premier élément et dont les n éléments suivants sont des listes représentant $F_1, ..., F_n$.

Par exemple, en prenant des étiquettes de type entier, l'arbre élémentaire A_1, comportant un unique sommet (qui n'a donc pas de fils) étiqueté par 1 sera représenté par la liste `(1)` comme l'indique la figure 1 ci-dessous.

Figure 1 : l'arbre élémentaire A_1 : représentation graphique et représentation sous forme de liste

La figure 2 donne l'exemple d'un arbre étiqueté, A, non élémentaire :

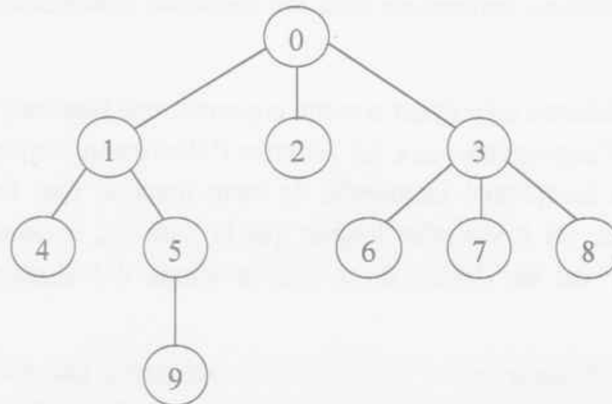

Figure 2 : l'arbre A, représenté par `(0 (1 (4) (5 (9))) (2) (3 (6) (7) (8)))`

Remarque : Cette définition exclut la notion d'arbre vide, c'est-à-dire avec zéro sommet.

Dans la suite de ce sujet, on ne considèrera que des arbres étiquetés, comme définis ci-dessus, qui seront désignés, pour simplifier, par le terme "arbre". Par abus, on pourra assimiler les sommets à leurs étiquettes, et dire, par exemple pour l'arbre A de la figure 2, que "le sommet 3 a trois fils, 6, 7, et 8". On appellera *feuille* tous les sommets qui n'ont pas de fils. Par exemple, les feuilles de l'arbre A de la figure 2 sont les sommets 4, 9, 2, 6, 7, et 8. On appelle *arête* l'arc qui "relie" un sommet et l'un de ses fils (on remarquera que cet arc n'apparaît ici que dans les représentations graphiques et qu'il n'est pas représenté explicitement dans les structures de données utilisées dans ce problème)

Première partie :

1°) La *hauteur* d'un sommet d'un arbre est définie comme le nombre d'arêtes du chemin qui relie ce sommet à la racine de l'arbre. Par exemple, dans l'arbre A de la figure 2, le sommet 0 est de hauteur 0, les sommets 1, 2 et 3 de hauteur 1, les sommets 4, 5, 6 et 8 de hauteur 2, etc. La hauteur d'un arbre est définie comme le maximum des hauteurs de ses sommets. La hauteur de l'arbre A de la figure 2 est donc la hauteur du sommet 9, c'est-à-dire 3.

Ecrire une fonction Scheme `hauteur` ayant comme argument un arbre `A` et telle que l'évaluation de l'expression `(hauteur A)` retourne la hauteur de l'arbre `A`.

2°) Ecrire une fonction Scheme `feuilles` ayant comme argument un arbre `A` et telle que l'évaluation de l'expression `(feuilles A)` retourne la liste des feuilles de `A`.

3°) Ecrire une fonction Scheme `hauteurSommet` ayant comme arguments un arbre `A` et un sommet `s` de `A` et telle que l'évaluation de l'expression `(hauteurSommet A s)` retourne la hauteur du sommet `s`.

Indication : Dans la liste représentant `A`, la hauteur du sommet `s` est donnée par son "niveau de profondeur" dans les sous-listes. Par exemple : 0 n'est dans aucune sous-liste, et sa "profondeur" est 0 ; 1 est dans une sous-liste, et sa "profondeur" est 1 ; 4 est dans une "sous-sous-liste", et sa "profondeur" est 2 ; etc.

Deuxième partie :

1°) Un arbre sera dit *partiellement équilibré* s'il existe un entier n tel que tous ses sous-arbres fils sont de hauteur n ou $n - 1$. Par exemple, l'arbre A de la figure 2 n'est pas (partiellement) équilibré parce que les hauteurs de ses sous-arbres fils sont respectivement 2, 0 et 1.

L'arbre A', représenté sur la figure 3 ci-dessous est partiellement équilibré parce que ses sous-arbres sont respectivement de hauteurs 2 et 3.

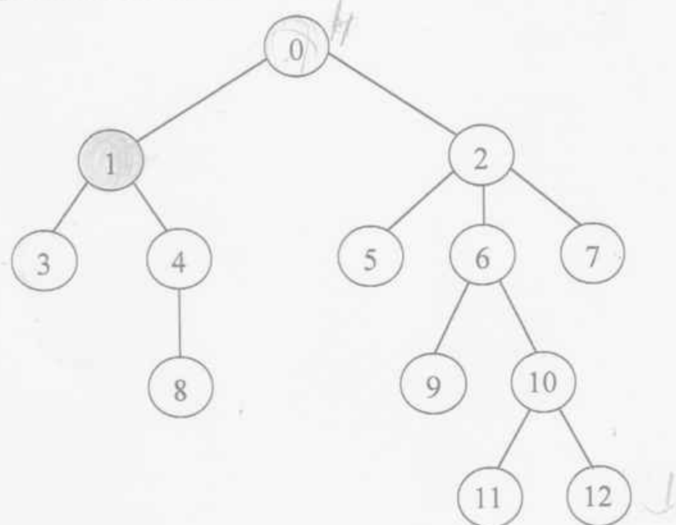

Figure 3 : l'arbre A' est un arbre partiellement équilibré

Ecrire une fonction Scheme `partiellementEquilibré` ayant comme argument un arbre `A` et telle que l'évaluation de l'expression `(partiellementEquilibré A)` retourne `#t` si `A` est partiellement équilibré et `#f` sinon.

Remarque : Attention ! Le nombre de sous-arbres est quelconque.

2°) Un arbre sera dit *totalement équilibré* s'il existe un entier n tel que toutes ses feuilles sont de hauteur n ou $n - 1$. Par exemple, l'arbre A' de la figure 3 n'est pas totalement équilibré parce qu'il existe des feuilles de hauteur 2, 3 et 4.

Ecrire une fonction Scheme `totalementEquilibré` ayant comme argument un arbre `A` et telle que l'évaluation de l'expression `(totalementEquilibré A)` retourne `#t` si `A` est totalement équilibré et `#f` sinon.

Remarque : Attention ! Le nombre de sous-arbres est quelconque.